How to Use SAM with ROOT Files

Reiner Hauser, rhauser@fnal.gov v0.1, 29 October 2004

This document describes how to access ROOT files stored in SAM from an analysis program.

Contents

1 Introduction

This document describes how to access ROOT files stored in SAM from an analysis program. It expands and updates the original documentation that can be found *here* <http://d0db-prd.fnal.gov/sam/doc/userdocs/SamRoot.html> .

Due to the large number of different analysis environments, there can be no simple re-usable solution. The assumption here is that the analysis program is a compiled C++ executable (not a ROOT macro). It may or may not use the D0 build system. In either case the necessary libraries can be easily included.

2 Getting Started

2.1 Which Release to use

The following assumes a release using GCC 3.1. The necessary UPS package is called `sam_client_api`, so you should check which versions are available:

```
% ups list -aK+ sam_client_api
"sam_client_api" "v4_2_1_20" "Linux+2.4" "GCC-3.1" "current"
```

2.2 Setting up the Release

```
% setup sam_client_api -q GCC-3.1 [1]
% setup sam                       [2]
```

In addition, you should do any setups for the D0 release you normally need, either for the full release, or the correct ROOT version etc.

3 An Example Program

As a first try you can copy over the demo directory and compile and run it.

```
cp -r $SAM_CLIENT_API_DIR/demo .
cd demo
```

Most likely, the provided GNUmakefile will not work. Here is a version that uses whatever ROOT version you have setup before:

```
#
# Adapted from ${SAM_CLIENT_API}/demo/GNUmakefile
#

CXXFLAGS :=
LDFLAGS  :=

ifeq (GCC, $(findstring GCC, $(BFARCH)))
  CXX := g++
  LD := g++

  THREADS := -lpthread
  CXXFLAGS +=  -DHAVE_JTC_STOP
endif

APPLIBS := -lsam_corba -lsam_util -lOB -lCosNaming -lJTC -lm $(THREADS)


CXXFLAGS += -I$(ORBACUS_DIR)/include/ \
 -I$(SAM_CLIENT_API_DIR)/ \
 -I$(SAM_LIB_DIR)/

### Flags to link the library statically
LDFLAGS += -L$(ORBACUS_DIR)/lib  -L$(SAM_LIB_DIR)/lib \
            -L$(SAM_CLIENT_API_DIR)/lib

### Flags to link the library dynamically
#LDFLAGS += -L$(ORBACUS_DIR)/lib  -L$(SAM_LIB_DIR)/lib/so \
```

```
#           -L$(SAM_CLIENT_API_DIR)/lib/so

LOADLIBES := -lsam_client_api $(APPLIBS)

CPPFILES := APIdemo.cpp
OBJFILES := $(patsubst %.cpp,$(tmpdir)%.o,$(CPPFILES))
DFILES := $(OBJFILES:.o=.d)

# Get root definitions
ROOTLDFLAGS = $(shell root-config --ldflags)
ROOTCFLAGS  = $(shell root-config --cflags)
ROOTLIBS    = $(shell root-config --libs)
ROOTINCLUDE = $(shell root-config --incdir)

CXXFLAGS  += -I$(ROOTINCLUDE)
LDFLAGS   += $(ROOTLDFLAGS)
LOADLIBES += $(ROOTLIBS)

vpath %.cpp
vpath %.o

vpath %.cpp $(addprefix $(tmpdir),$(idl_packages)) $(shell pwd) $(tmpdir)
vpath %.o $(addprefix $(tmpdir),$(idl_packages))


BINARY   := $(bindir)APIdemo

$(bindir)APIdemo: $(OBJFILES)
        @echo "*** L *** $(notdir $@)"
        $(CXX) $(LDFLAGS)  -o $@ $^ $(LOADLIBES)

$(tmpdir)%.d:%.cpp
        @echo "*** D *** $(notdir $@)"
        @$(SHELL) -ec "$(CXX) $(CXXFLAGS) -M $< \
                       | sed 's?..*\.o:?$(tmpdir)&?g' >  $@"
$(tmpdir)%.o:%.cpp
        @echo "*** C *** $(notdir $@)"
        $(CXX) $(CXXFLAGS) -o $@ -c $<

all: demo
demo: $(BINARY)

-include $(DFILES)
```

If you type `make`, you should get an executable `APIdemo`.

You can submit this, e.g. on `clued0` like this:

```
sam submit job --defname=NPtree-2MU-p14.06.00 \
               --cpu-per-event=1s \
               --script=APIdemo \
               --group=dzero
```

Here is a modified copy of the original `APIdemo.cpp`. It allows to specify a maximum number of files to process.

```
#include <time.h>
#include "sam_client_api/SAMcppClientAPI.hpp"

#include "TFile.h"
#include "TTree.h"

using namespace std;

int main(int argc, char *argv[])
{
 int num_files = 99999999;

 if (argc > 1) {
     num_files = strtol(argv[1], 0, 0);
     if (num_files == 0) {
         cerr << "Invalid argument for num_files: " << argv[1] << endl;
         exit(1);
     }
 }

 sam::SAMAPIOpts myOpts;
 myOpts.setAppName("demo");
 myOpts.setAppVersion("1");
 myOpts.setWorkGroup("demo");

 int files_read = 0;

 try {
   sam::SAMcppClientAPI myAPI(& myOpts);
   while(files_read < num_files && myAPI.nextInputFile()) {

     cout << "API: Got file" << myAPI.currentInputFile() << endl;
     TFile file(myAPI.currentInputFile().c_str());
     TTree *tree = (TTree*)file.Get("TMBTree");

     Int_t entries = (Int_t) tree->GetEntries();

     cout << "File has " << entries  << " events " << endl;
```

```
    time_t start = time(0);
    for(Int_t i = 0; i < entries; i++) {
        tree->GetEntry(i);
    }
    cout << "API: Done with file " << myAPI.currentInputFile() << endl;
    cout << "Took: " << time(0) - start << " seconds" << endl;

    files_read++;
  }
 }
 catch(sam::Exception& ex) {
   cout << ex.error() << endl;
   exit(1);
 }
 catch(...) {
   cout << "Error in the API" << endl;
   exit(1);
 }

 // here you can display your histogram or use the TChain.

 cout << "Exiting..." << endl;
}
```

Note that this program assumes that the input files are in `TMBTree` format and searches for a `TTree` with this name. It should be straightforward to adapt it to any other format.

3.1 Adapting the Library/Example

Obviously, the example program provides an outer loop over all files in the SAM dataset. In principle, you can insert your analysis code where the `GetEntry()` method of the `TTree` is called. In practice it might be some more work to incorporate it into your existing program structure.

However, the important things to remember are:

- The SAM project is started outside the application.
- You have to set at a few values in the `SAMAPIOpts` variable. See the SAMRoot page for full details. Here we used `demo` as the application name, but you can register your know application name and version by sending an e-mail to *d0sam-admin@fnal.gov* <mailto:d0sam-admin@fnal.gov>
- You have to instantiate the `sam::SAMcppClientAPI` class and pass it the options.
- Then repeatedly call `nextInputFile()` on this object until it returns false.
- Call `currentInputFile()` to get the next file name. Open it and do whatever you do in your analysis...

Note that you can't use a `TChain` instead of a `TTree`. Each file is delivered to you by SAM as you process them. For any non-trivial dataset, the local cache of your execution host (e.g. CAB) will not be able to keep all the files anyway.

4 Submitting Jobs to CAB

When running over SAM datasets, you would like to use not only `clued0`, but also `CAB`. In addition you would like to run parallel jobs, just like you do it with the D0 framework and `d0tools`.

You can find several scripts to help you with this at `/work/theriv-clued0/rhauser/sam_cab`. In the following we call this directory `$CABSUB`.

There is a submission tool `submit.job` similar to `rund0exe`, but for generic executables. So it does not require you to specify RCPs etc and will not complain if it doesn't find things in the usual place.

```
% ${CABSUB}/submit.job -h
usage: ./submit.job [ options ]
where options are:
   -help|-h|--help          print this help
   -dest=<DESTINATION>      submit job to destination [ required ]
                            currently: clued0, cab, cabsrv1
   -clued0                  short cut for -dest=clued0
   -cab                     short cut for -dest=cab
   -cabsrv1                 short cut for -dest=cabsrv1
   -def=<DEFINITION>        SAM dataset definition [ required ]
   -project=<NAME>          SAM project name [ default: ${USER}_[date] ]
   -jobs=<NUMBER>           number of parallel SAM jobs [ default: 1 ]
   -exec=<EXECUTABLE>       name of executable [ default: APIdemo ]
   -args=<ARGUMENTS>        arguments passed to executable [ default: none ]
   -name=<NAME>             name of job in batch system [ default: run_jobs.sh ]
   -outhost=<HOSTNAME>      name of output host [ default: hostname ]
   -outdir=<DIRECTORY>      name of output directory [ default: pwd ]
   -inputdir=<DIRECTORY>    name of input directory [ default: pwd ]
   -queue=<QUEUE>           name of queue in batch system
```

You pass it a destination (one of `clued0`, `cab` and `cabsrv1`), an executable and a SAM definition name. It will execute the job on the given cluster. In addition, it accepts parameters similar to `d0tools`. Their meaning should be obvious from the description.

The `-args=` parameter can be used to pass arbitrary arguments to the executable. If there is more than one, they have to be put into quotes, so you can separate them by spaces.

```
${CABSUB}/submit.job -dest=cabsrv1 \
                     -def=NPtree-2MU-p14.06.00 \
                     -exec=APIdemo \
                     -outdir=/work/mymachine-clued0/scratch \
                     -jobs=10 \
                     -args=4
```

will run 10 jobs on `cabserver1`, each processing four input files from the given SAM dataset.